RL for Inventory Management

Progress Report - October 30, 2025

Group 4
Inara Hussain, Zihao Song, Hugo Cheng, Tasneem Soliman

1 MDP Framework

We use OR-Gym's inventory environment as the foundation, extended with external demand factors. A Markov Decision Process (MDP) formally defines the structure of the reinforcement learning problem. It specifies the information available to the agent (state space), the set of feasible decisions (actions), the probabilistic evolution of the system (transitions), and the resulting outcomes (rewards). In this framework, the agent learns to make optimal ordering decisions through repeated interaction with the simulated environment.

1.1 State Space ($\mathcal{S} \subseteq \mathbb{R}^{11}$)

$$s_t = (I_t, P_t, D_{t-1}, \ldots, D_{t-7}, W_t, M_t, \theta_t) \tag{1}$$

- $I_t \in [0, 500]$: On-hand inventory, representing the quantity of stock currently available for sale.
- $P_t \in [0, 600]$: Pipeline inventory, representing the quantity ordered but not yet received.
- $D_{t-i} \in [0, 200]$: Seven-day demand history, providing short-term temporal information about customer demand.
- $W_t \in \{0, 1, 2\}$: Weather condition (normal, adverse, or favorable), which influences demand levels by altering consumer behavior.
- $M_t \in \{0, 1\}$: Promotion indicator, capturing the occurrence of marketing campaigns that typically cause demand surges.
- $\theta_t \in [-1, 1]$: Seasonality variable (sine-encoded), representing recurring annual demand patterns.

Traditional inventory management models typically consider only current inventory and demand. Our approach extends this by integrating external contextual variables, such as weather, promotions, and seasonality. This allows the agent to anticipate demand fluctuations rather than react to them, which is the key innovation of our formulation.

1.2 Action Space

Discrete (for DQN): $\mathcal{A} = \{0, 10, 20, 30, 50, 75, 100, 150, 200\}$

The agent selects one of nine discrete order quantities, where $a_t = 0$ represents no order placement.

Continuous (for PPO, SAC): $\mathcal{A} = [0, 300]$

In this setting, the agent can issue any real-valued order quantity within the interval $[0, 300]$.

1.3 Transition Dynamics

The transition function describes how the system evolves in response to the agent's actions.

Inventory update (deterministic):

$$I_{t+1} = \max(0, I_t + A_{\text{arrived},t} - D_t) \tag{2}$$

$$P_{t+1} = P_t + a_t - A_{\text{arrived},t} \tag{3}$$

Inventory at time $t+1$ equals the current inventory plus any deliveries received minus customer demand. Negative inventory is not permitted; unmet demand corresponds to lost sales. Pipeline inventory tracks outstanding orders not yet fulfilled.

Demand generation (stochastic):

$$D_t \sim \text{Poisson}(\lambda_t) \tag{4}$$

$$\lambda_t = 50 \cdot (1 + 0.2 \cdot \mathbb{I}[W_t = 2] - 0.2 \cdot \mathbb{I}[W_t = 1] + 0.3 \cdot M_t + 0.15 \cdot |\sin(\theta_t)|) \tag{5}$$

Customer demand is modeled as a Poisson process with mean λ_t, which varies with weather, promotions, and seasonality.

Lead time is drawn from a discrete uniform distribution $L \sim \text{Uniform}(\{2, 3, 4, 5\})$, reflecting delivery delays of two to five days.

1.4 Reward Function

$$r_t = -\underbrace{1 \cdot I_t}_{\text{holding}} - \underbrace{10 \cdot \max(0, D_t - I_t)}_{\text{stockout}} - \underbrace{50 \cdot \mathbb{I}[a_t > 0]}_{\text{fixed cost}} - \underbrace{2 \cdot a_t}_{\text{variable cost}} \tag{6}$$

The reward represents the negative of total operational costs, including:

- Holding cost ($1/unit/day): Reflects the expense of storing excess inventory.
- Stockout penalty ($10/unit): Captures lost sales and customer dissatisfaction due to unmet demand.
- Fixed ordering cost ($50/order): Represents administrative or logistical expenses associated with placing an order.
- Variable cost ($2/unit): Denotes the unit purchasing cost.

The agent's objective is to minimize cumulative cost by balancing excessive inventory against the risk of stockouts. A discount factor $\gamma = 0.99$ ensures that long-term outcomes are valued nearly as much as immediate ones, encouraging proactive planning.

2 Algorithms

We evaluate four reinforcement learning algorithms, representing distinct methodological paradigms: value-based, policy-based, actor–critic, and model-based learning.

2.1 Double DQN (Deep RL Baseline)

Double DQN [1] addresses the overestimation bias present in standard DQN by decoupling action selection from action evaluation.

- **Type:** Value-based, learning a function $Q(s, a)$ estimating the expected return. **Architecture:** Two neural networks with two hidden layers (64 neurons each). One network chooses actions, the other evaluates them. This helps reduce overestimation.
- **Exploration:** ϵ-greedy schedule decaying from 1.0 to 0.01 over time.
- **Training:** 200,000 environment steps.

Listing 1: Double DQN

```
for episode in range(num_episodes):
    s = env.reset()
    for t in range(max_steps):
        a = epsilon_greedy(Q_net, s, epsilon)

        # Environment updates with contextual factors
        W_t = sample_weather()
        M_t = sample_promotion()
        D_t = sample_demand(W_t, M_t, _t)
        s_next, r, done, _ = env.step(a, W_t, M_t, D_t)

        replay_buffer.add(s, a, r, s_next, done)
        s = s_next

        if len(replay_buffer) > batch_size:
            batch = replay_buffer.sample(batch_size)
            target = r + gamma * Q_target(next_s, argmax_a Q_net(next_s))
            loss = (Q_net(s,a) - target)**2
            update(Q_net, loss)
        if done: break
    soft_update(Q_target, Q_net, tau)
```

Double DQN serves as the foundational deep RL baseline. The experience replay mechanism mitigates temporal correlation in training data, while the target network provides stable reference values during temporal difference updates.

2.2 Proximal Policy Optimization (PPO)

Proximal Policy Optimization [2] is an on-policy algorithm that maintains training stability through clipped probability ratios.

- **Type:** On-policy, policy-gradient method that directly optimizes the policy $\pi_\theta(a|s)$. **Architecture:** Actor-critic with shared layers. Two 64-neuron layers feed into:
  - Actor: Chooses the action.
  - Critic: Estimates how good the current state is.
- **Action space:** Continuous $[0, 300]$, allowing fine-grained order control.
- **Training:** 200,000 steps using Stable-Baselines3.

Listing 2: Proximal Policy Optimization (PPO)

```
for iteration in range(num_iterations):
    # Rollout collection with contextual sampling
    trajectories = []
    for step in range(horizon):
        W_t = sample_weather()
        M_t = sample_promotion()
        D_t = sample_demand(W_t, M_t, _t)
        a = policy.sample_action(s)
        s_next, r, done, _ = env.step(a, W_t, M_t, D_t)
        trajectories.append((s, a, r, s_next, done))
        s = s_next
        if done: s = env.reset()

    advantages = compute_GAE(trajectories, value_net, gamma, lambda_)

    for epoch in range(num_epochs):
        for batch in minibatch(trajectories):
            ratio = policy(a|s) / old_policy(a|s)
            clip_obj = min(ratio*A, clip(ratio, 1-eps, 1+eps)*A)
            loss = -mean(clip_obj) + vf_coef*MSE(V(s), R) - ent_coef*entropy
            update(policy, loss)
```

PPO maintains training stability through constrained policy updates and improved advantage estimation. Its ability to handle continuous actions makes it well-suited for inventory control tasks requiring smooth order adjustments.

2.3 Soft Actor-Critic (SAC)

Soft Actor-Critic [3] combines off-policy learning with maximum entropy reinforcement learning to encourage exploration.

- **Type:** Off-policy actor–critic algorithm optimizing both reward and policy entropy. **Architecture:**
  - Twin critics: Two critic networks (256 neurons each) give conservative estimates to avoid overconfidence.
  - Gaussian policy: Actor outputs a distribution over actions, not just a single choice.
- **Action space:** Continuous $[0, 300]$.
- **Training:** 200,000 steps using Stable-Baselines3.

Listing 3: Soft Actor-Critic (SAC)

```
for step in range(total_steps):
    # Sample contextual environment variables
    W_t = sample_weather()
    M_t = sample_promotion()
    D_t = sample_demand(W_t, M_t, _t)

    a = policy.sample_action(s)
    s_next, r, done, _ = env.step(a, W_t, M_t, D_t)
    replay_buffer.add(s, a, r, s_next, done)
    s = s_next

    if step > warmup:
        batch = replay_buffer.sample(batch_size)
        y = r + gamma * (min(Q1', Q2') - alpha * log_pi)
        update(Q1, (Q1 - y)**2)
        update(Q2, (Q2 - y)**2)
        policy_loss = alpha*log_pi - min(Q1, Q2)
        update(policy, policy_loss)
        adjust_temperature(alpha)
        soft_update(target_Q, Q, tau)
```

SAC encourages exploration by maximizing entropy while learning the optimal policy. The dual critic design reduces Q-value overestimation bias and offers excellent sample efficiency for continuous control.

2.4 Dyna-Q (Model-Based Reinforcement Learning)

Dyna-Q [4] integrates model-free learning with model-based planning by learning an environment model and using it to generate simulated experience.

- **Type:** Hybrid model-based and model-free approach combining Q-learning with learned environment dynamics.
- **Components:** Q-function updated using both real and simulated experiences generated from learned transition and reward models.
- **Model:** Neural networks with 128-neuron hidden layers predicting next state and reward.
- **Planning:** Ten simulated updates per real experience step, enhancing sample efficiency.
- **Training:** 50,000 episodes.

Listing 4: Dyna-Q

```
for episode in range(num_episodes):
    s = env.reset()
    for t in range(max_steps):
        a = epsilon_greedy(Q, s, epsilon)

        # Environment dynamics with external factors
        W_t = sample_weather()
        M_t = sample_promotion()
        D_t = sample_demand(W_t, M_t, _t)
        s_next, r, done, _ = env.step(a, W_t, M_t, D_t)

        Q.update(s, a, r, s_next)
        model.update(s, a, r, s_next)

        # Simulated planning using learned model
        for _ in range(n_planning_steps):
            s_p, a_p = model.sample_experience()
            r_p, s_next_p = model.predict(s_p, a_p)
            Q.update(s_p, a_p, r_p, s_next_p)

        s = s_next
        if done: break
```

Dyna-Q learns an internal model of the environment and uses it to generate synthetic experience, effectively extending the agent's training data. This integration of model-based planning with model-free learning improves efficiency and interpretability.

3 Novel Contribution

Building on Ferretti and Marchi (2024), our work introduces external demand drivers such as weather, promotions, and seasonality into the state representation. These variables induce non-stationary demand patterns and enable proactive inventory decisions. Whereas prior methods responded reactively to past demand, our approach allows anticipation of future shifts. For example, the agent can increase orders before an upcoming promotion or favorable weather period.

To validate this contribution, we will conduct an ablation study comparing:

- Baseline model: (I_t, P_t, D_t) only.
- Enhanced model: $(I_t, P_t, D_t, W_t, M_t, \theta_t)$ with contextual features.

Performance differences will quantify the value added by incorporating external information, measured through total cost reduction and improved policy stability.

4 Individual Contributions

Phase 1

For the first phase of the project (September 18 - October 15), team members contributed as follows:

- **Inara Hussain and Zihao Song:** Conducted extensive research on reinforcement learning algorithms suitable for inventory management, including Double DQN, PPO, SAC, and Dyna-Q. Prepared and wrote this progress report, including the MDP formulation, algorithm descriptions, and development plan.
- **Hugo Cheng and Inara Hussain:** Implemented the custom inventory environment extending OR-Gym, including the integration of external demand factors (weather, promotions, seasonality), stochastic demand generation, and environment dynamics. Developed visualization utilities for monitoring agent performance.
- **Tasneem Soliman:** Set up the project GitHub repository, established version control workflows, and created the initial project structure for collaborative development.

Note: Christopher Delaney is no longer part of the group as of October 15, 2025. The remaining four members will continue with the project.

Phase 2

- **Inara Hussain**
  - Added evaluation framework to display different RL algorithms and compare `[3f01bae]`
  - Updated progress report `[b2d6a44]`
- **Zihao Song**
  - removed .idea and __pycache__ and added .gitignore `[5c926f8]`
  - Added QLearning and statediscretizer `[119a1ec]`
  - Edited group proposal submission video
- **Tasneem Soliman**
  - Updated ExtendedInventoryEnv `[0c02e6c]`
- **Hugo Cheng**
  - Created slides and script for env demo video

5 Phases

5.1 Phase 1 (Sept 18 - Oct 15): Environment Setup [DONE]

- Extended OR-Gym environment with weather and promotion features.
- Implemented stochastic demand generator with contextual modifiers.
- Created visualization and evaluation utilities.
- Established repository structure and testing framework.
- Decided on algorithms being used

Status: Completed. The extended environment accurately simulates inventory dynamics under variable conditions.

5.2 Phase 2 (Oct 16 - Oct 30): Baseline [DONE]

- Implement tabular Q-learning with discretized states.
- Implement heuristic baselines (base-stock and (s, S) policies).
- Train Q-learning for 50,000 episodes.
- Evaluate performance relative to heuristics.
- Deliverable: Baseline results and demonstration video.

Objective: Establish a benchmark using simple RL and heuristic methods against which deep RL methods can be evaluated.

5.3 Phase 3 (Oct 31 - Nov 13): Deep RL

- Implement Double DQN (PyTorch) and configure PPO and SAC (Stable-Baselines3).
- Train all algorithms for 200,000 steps.
- Perform hyperparameter tuning.
- Compare algorithmic performance.
- Deliverable: Comparative analysis and demonstration video.

Objective: Evaluate deep reinforcement learning algorithms and determine the most effective approach for this environment.

5.4 Phase 4 (Nov 14 - Nov 27): Advanced Analysis

- Implement Dyna-Q with a learned transition and reward model.
- Conduct ablation study comparing baseline and enhanced state representations.
- Retrain top-performing algorithms on all variants.
- Perform statistical significance testing (t-tests with Bonferroni correction).
- Conduct feature importance analysis.
- Deliverable: Comprehensive experimental results and feature impact summary.

Objective: Assess the impact of contextual features and validate results statistically.

5.5 Phase 5 (Nov 28 - Dec 4): Finalization

- Write final report and finalize analysis.
- Ensure reproducibility (README, requirements.txt, random seeds).
- Code cleanup and documentation.
- Generate visualizations and summary tables.
- Address feedback from prior evaluations.

Objective: Finalize the project with complete documentation, reproducible code, and well-structured results for submission.

6 References

References

[1] van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

[2] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[3] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2019). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

[4] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4), 160-163.